Kubernetes Operations

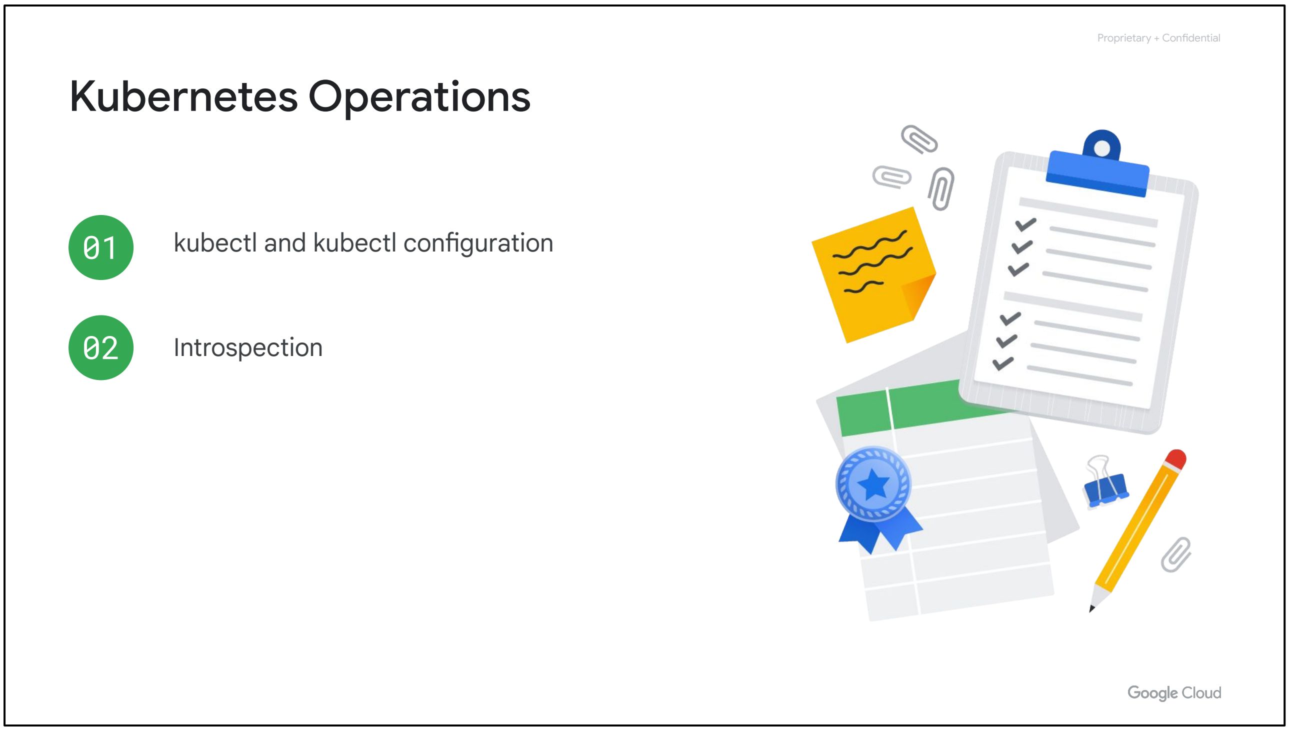
04

Google Cloud

SAY: **kubectl** is a command-line tool to interact with GKE clusters. Because it allows you to manage Kubernetes resources from the command line, it makes it easy to automate tasks and to troubleshoot problems.

But how does it work? Does it need special configuration? How can it be used to gather information about the containers, Pods, services, and other engines that run within the cluster?

The goal of this final section of the course, titled “Kubernetes Operations” was designed to answer those questions.



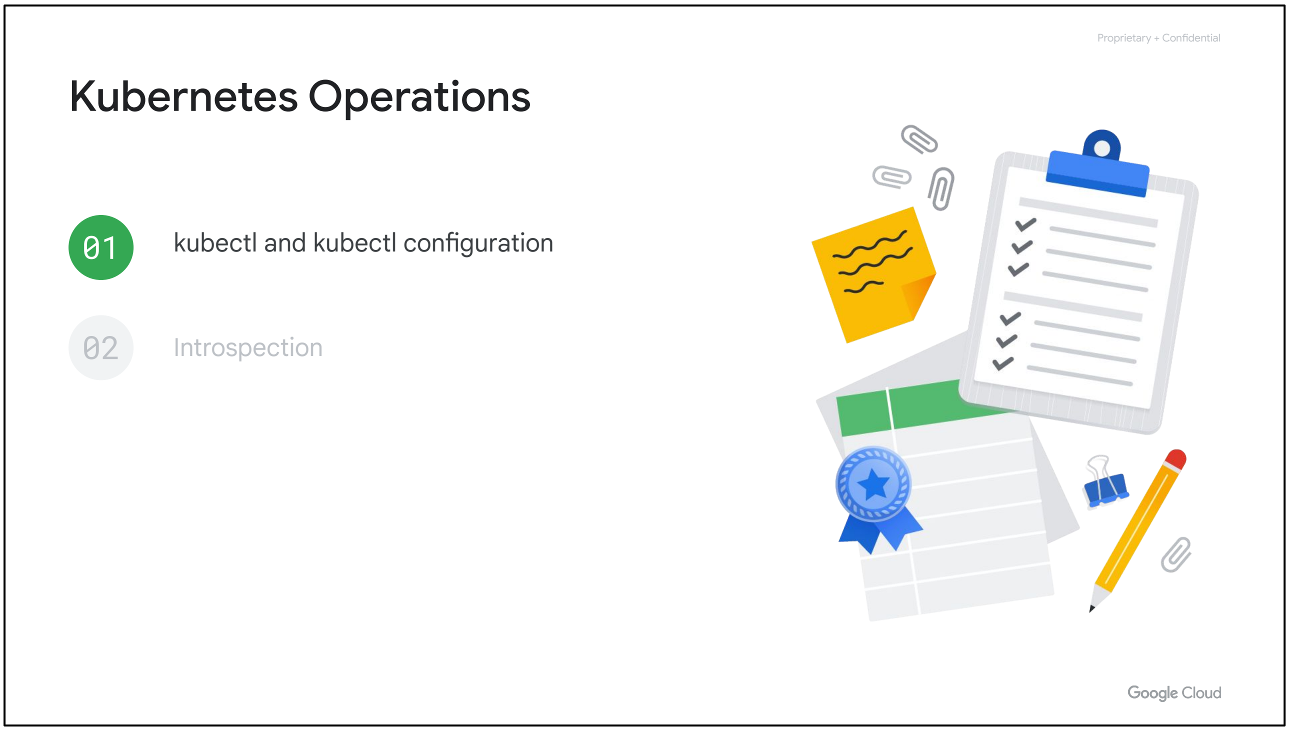
Kubernetes Operations

01 kubectl and kubectl configuration

02 Introspection

SAY: You explore kubectl and how to configure it, and what introspection means and how it can be used to troubleshoot a cluster.

You also get hands on practice deploying Google Kubernetes Engine clusters from Cloud Shell.



Proprietary + Confidential

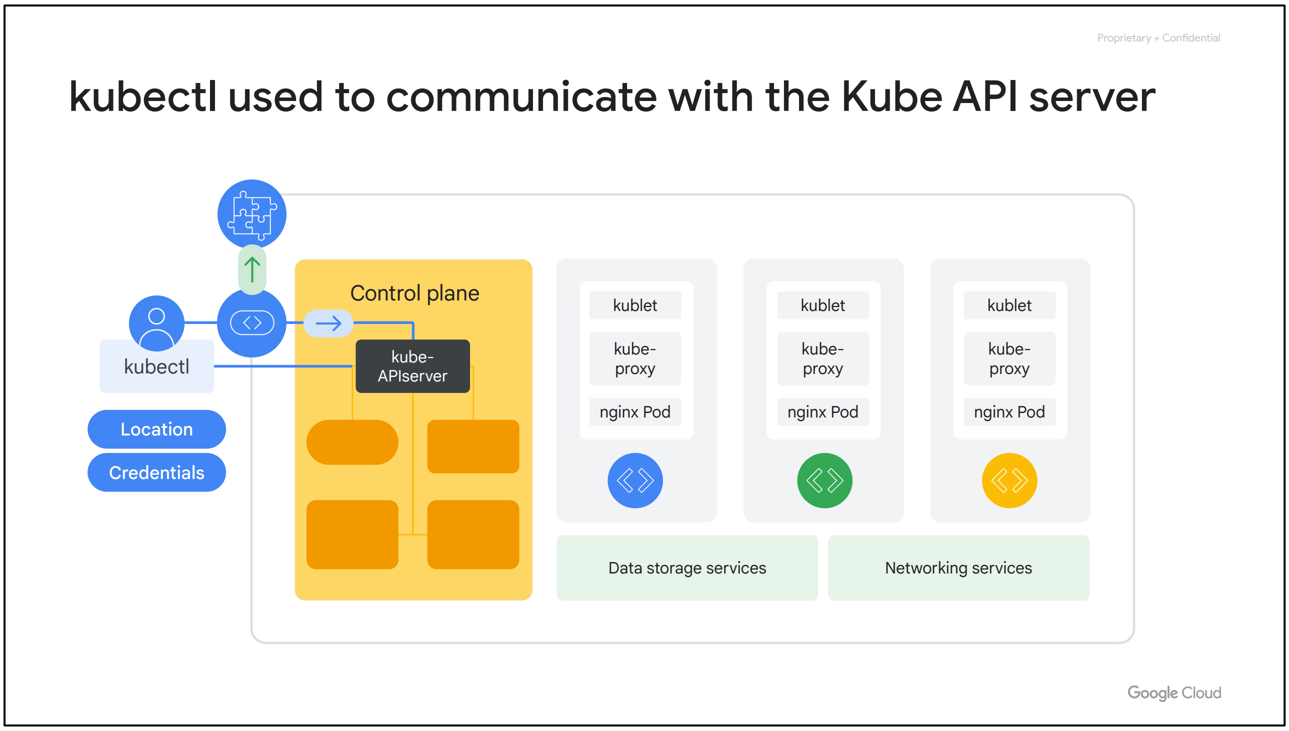
Kubernetes Operations

01 kubectl and kubectl configuration

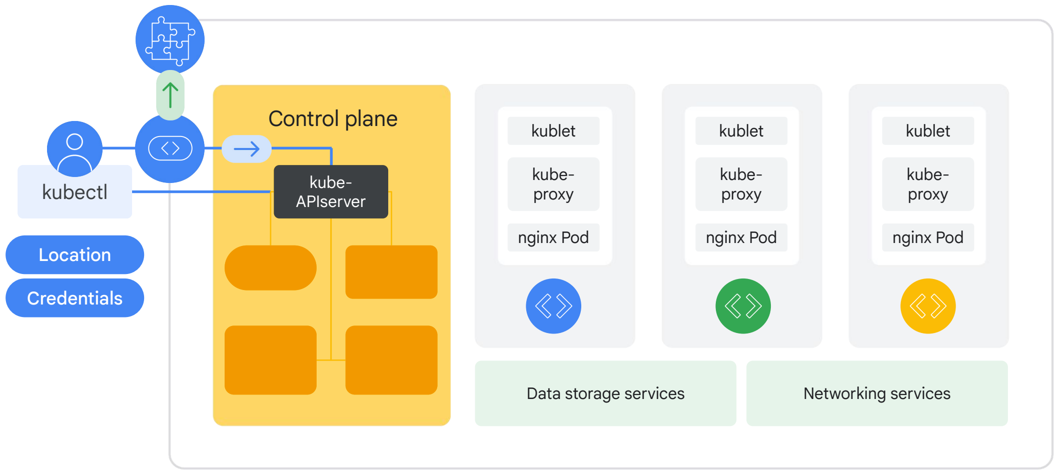
02 Introspection

Google Cloud

SAY: So, what exactly is the **kubectl command** and why is it important?



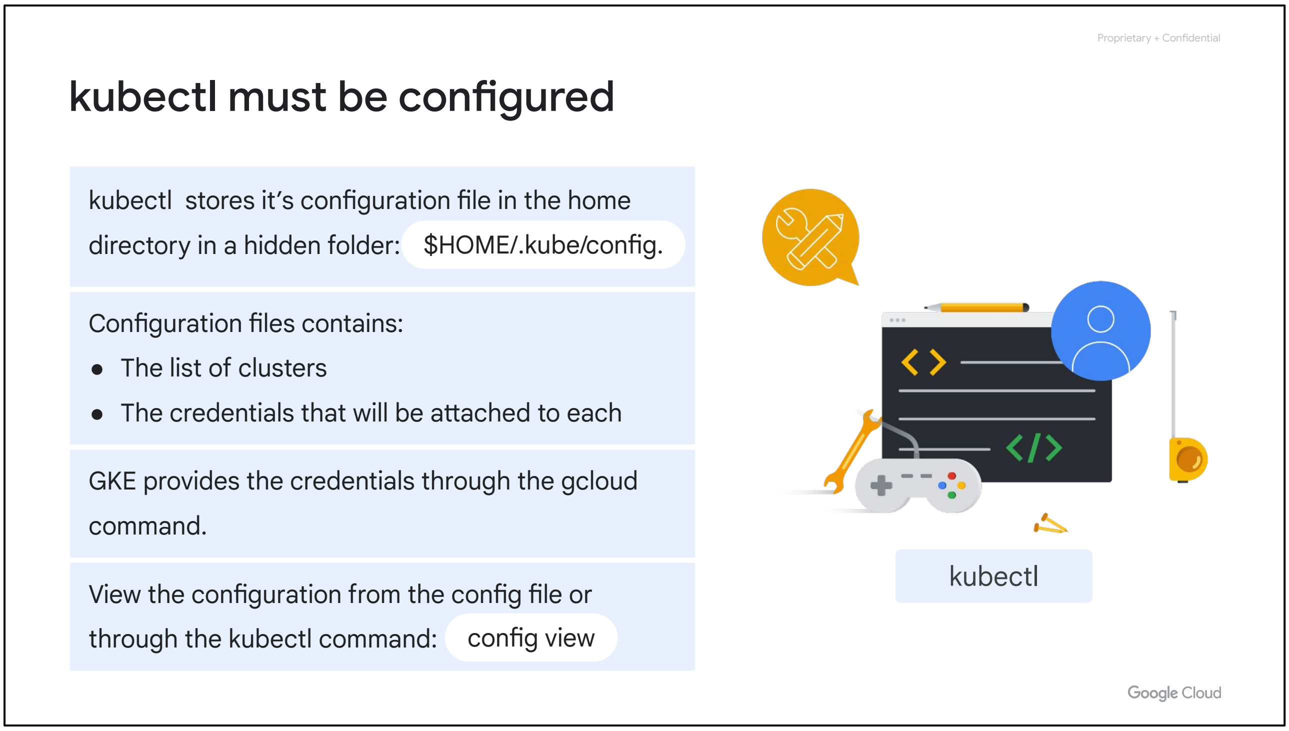
kubectl used to communicate with the Kube API server



SAY: It's used to communicate with the Kube API server on the control plane.

This is important to users, because it allows them to make requests to the cluster and kubectl determines which part of the control plane to communicate with.
Within a selected Kubernetes cluster, kubectl transforms command-line entries into API calls and sends them to the Kube API server.

However, to work properly, kubectl must be configured with the location and credentials of a Kubernetes cluster.



kubectl must be configured

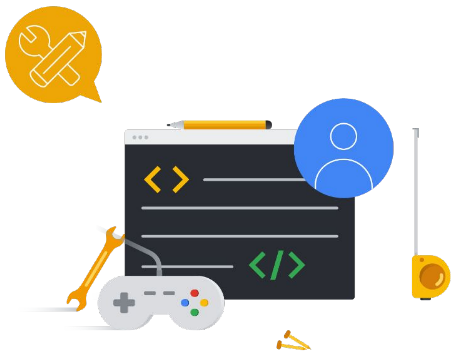
kubectl stores it's configuration file in the home directory in a hidden folder: `$HOME/.kube/config.`

Configuration files contains:

- The list of clusters
- The credentials that will be attached to each

GKE provides the credentials through the gcloud command.

View the configuration from the config file or through the kubectl command: `config view`

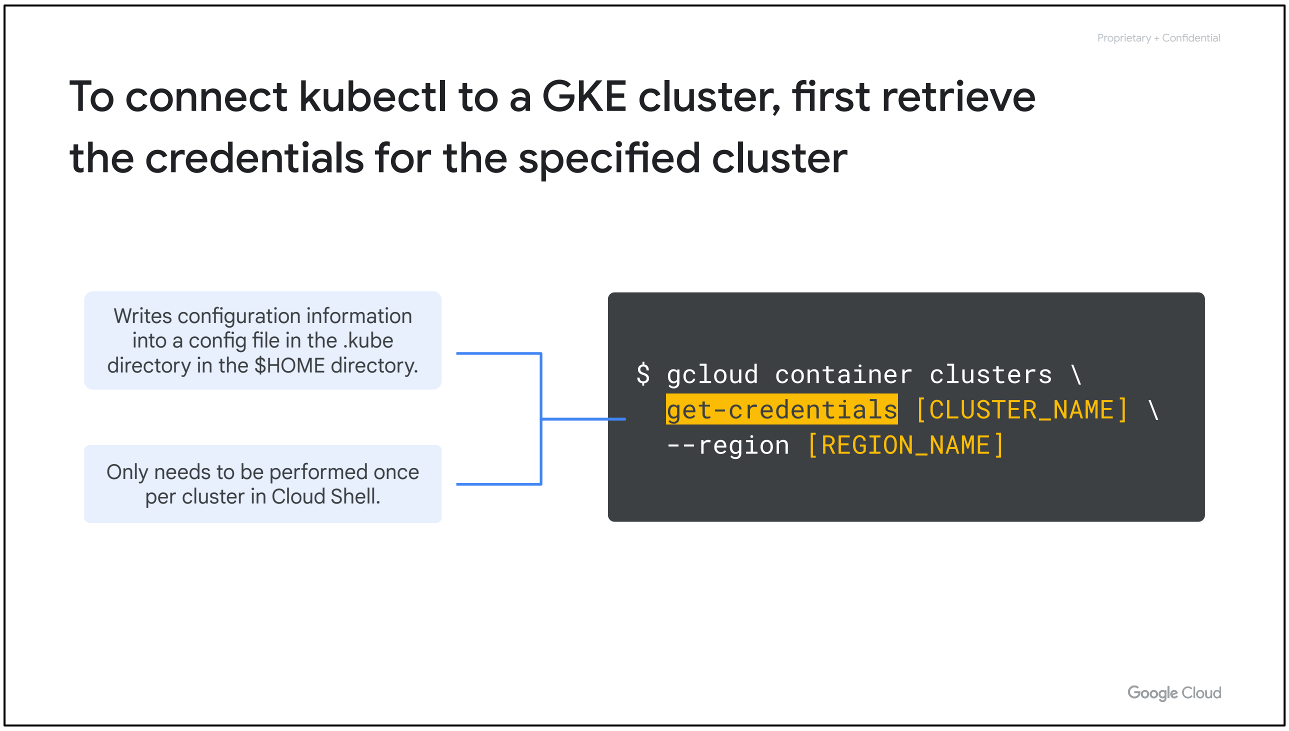


SAY: And before kubectl can be used to configure a cluster, it must *first* be configured.

kubectl stores its configuration in a file in the home directory in a hidden folder named .kube, and contains the list of clusters and the credentials that will be attached to each of those clusters.

But where do the credentials come from? GKE provides them through the gcloud command. To view the configuration, either open the config file or use the kubectl command: "config view".

Please note that the kubectl config shows the configuration of the kubectl command itself, whereas other kubectl commands show the configurations of cluster and workloads.



Proprietary + Confidential

To connect kubectl to a GKE cluster, first retrieve the credentials for the specified cluster

Writes configuration information into a config file in the .kube directory in the $HOME directory.

Only needs to be performed once per cluster in Cloud Shell.

```
$ gcloud container clusters \
  get-credentials [CLUSTER_NAME] \
  --region [REGION_NAME]
```

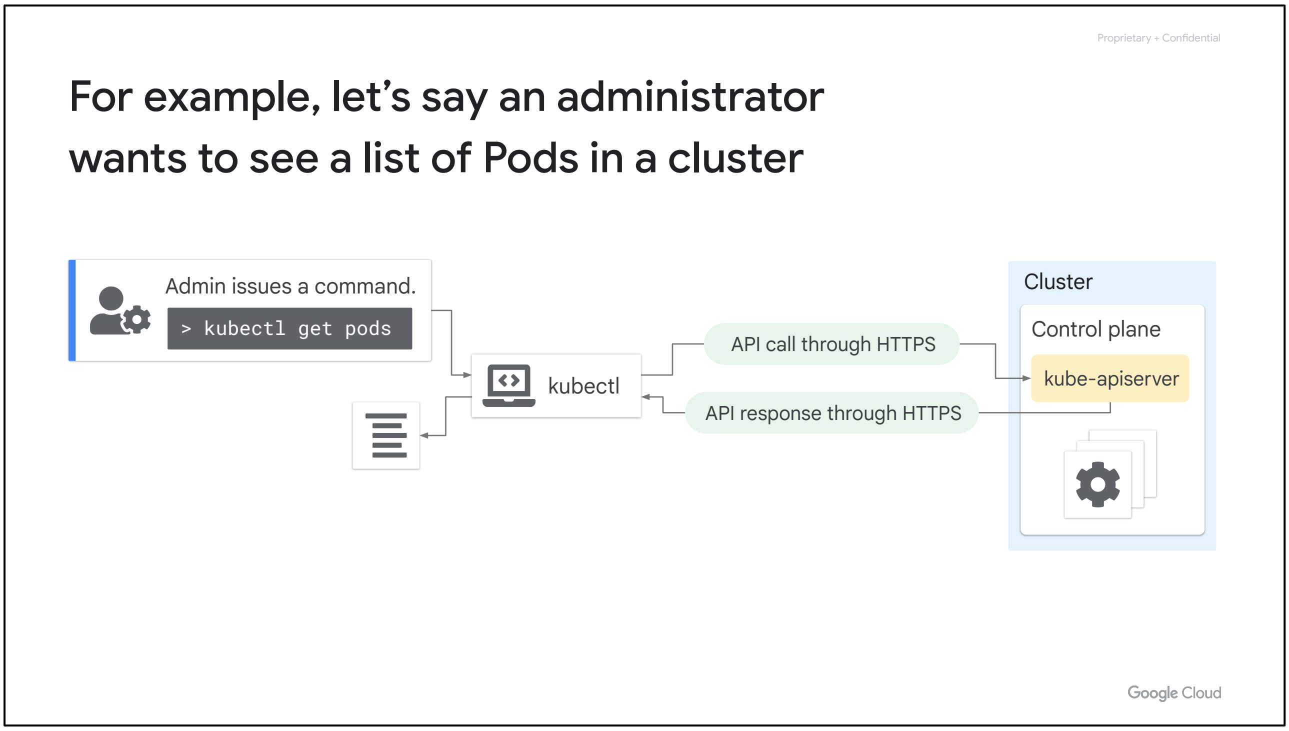
Google Cloud

SAY: To connect kubectl to a GKE cluster, first retrieve the credentials for the specified cluster.

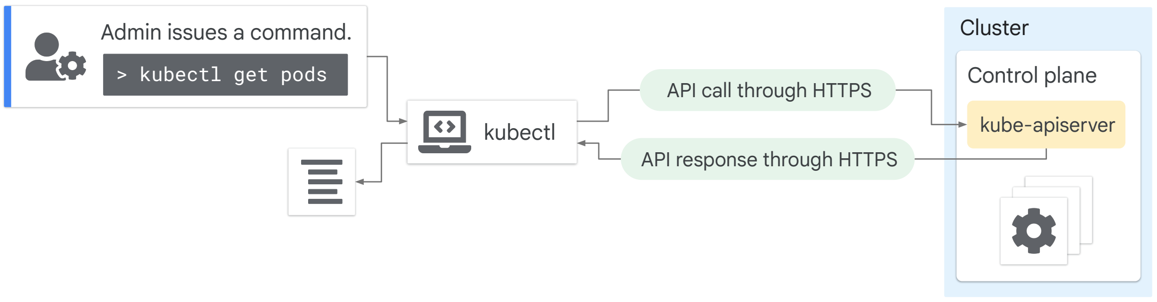
This can be done with the “get-credentials” gcloud command in any other environment where the gcloud command-line tool and kubectl are installed (note that both are installed by default in Cloud Shell).

By default, the gcloud “get-credentials” command writes configuration information into a config file in the .kube directory in the $HOME directory. If this command is rerun for a different cluster, it’ll update the config file with the credentials for the new cluster. This configuration process only needs to be performed once per cluster in Cloud Shell, because the .kube directory and its contents stay in the $HOME directory.

The gcloud command is how authorized users interact with Google Cloud from the command line. If authorized, the gcloud “get-credentials” command provides the credentials needed to connect with a GKE cluster.



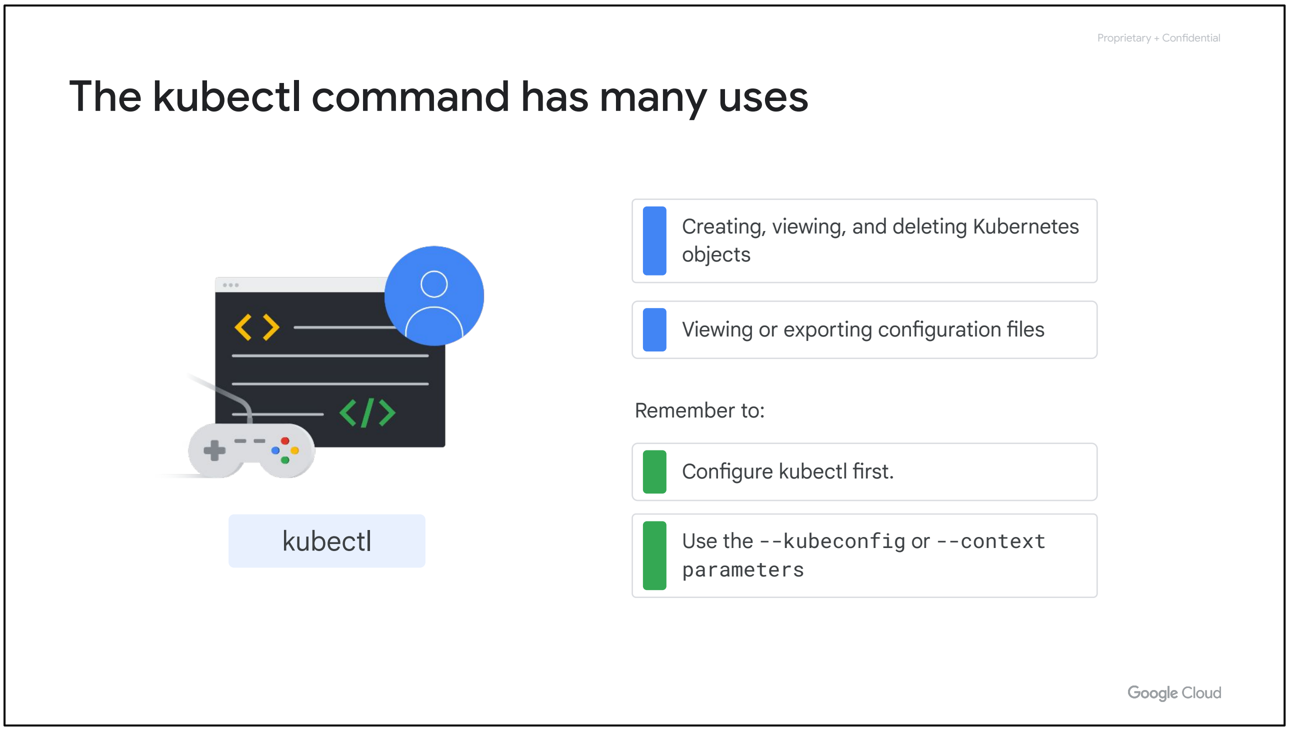
For example, let's say an administrator wants to see a list of Pods in a cluster



```
> kubectl get pods
```

SAY: For example, let's say an administrator wants to see a list of Pods in a cluster.

- After connecting kubectl to the cluster with proper credentials, the administrator can issue the kubectl "get pod" command.
- kubectl then converts this command into an API call, which it sends to the Kube API server through HTTPS on the cluster's control plane server.
- From there, the Kube API server processes the request by querying etcd.
- The Kube API server then returns the results to kubectl through HTTPS.
- Finally, kubectl interprets the API response and displays the results to the administrator at the command prompt.



The kubectl command has many uses

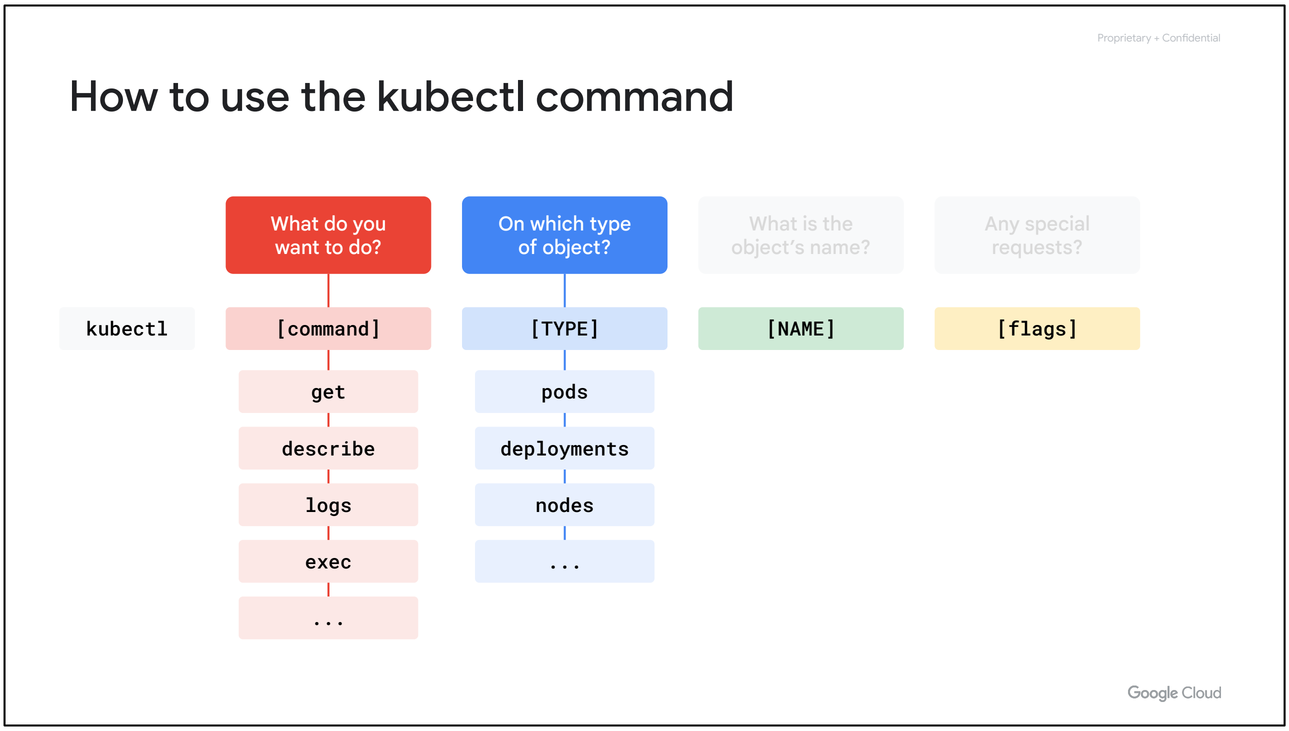
- Creating, viewing, and deleting Kubernetes objects
- Viewing or exporting configuration files

Remember to:

- Configure kubectl first.
- Use the `--kubeconfig` or `--context` `parameters`

SAY: The kubectl command has many uses, from creating Kubernetes objects, to viewing them, deleting them, and viewing or exporting configuration files.

Just remember to configure kubectl first or to use the `--kubeconfig` or `--context` `parameters`, so that the commands you type are performed on the cluster you intended.



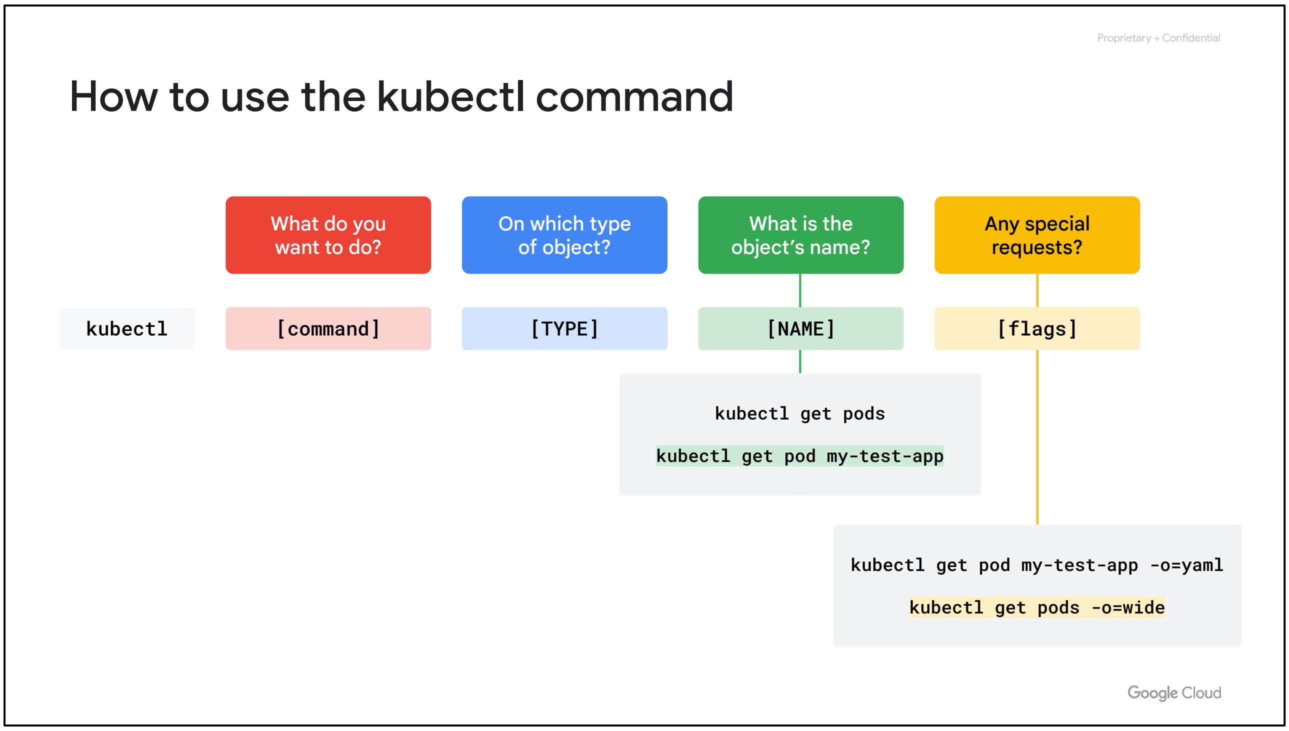
How to use the kubectl command

SAY: Now let's explore how to use the kubectl command. kubectl's syntax is composed of four parts: the command, the type, the name, and optional flags.

The **command** specifies the action that you want to perform, such as get, describe, logs, or exec. Some commands show information, whereas others change the cluster's configuration.

The **TYPE** defines the Kubernetes object that the "command" acts upon, like Pods, deployments, nodes, or other objects, including the cluster itself.

The TYPE used with a "command" tells kubectl what you want to do and the type of object you want to perform that action on.


Proprietary + Confidential

How to use the kubectl command

```
kubectl [command] [TYPE] [NAME] [flags]
```

```
kubectl get pods
kubectl get pod my-test-app
```

```
kubectl get pod my-test-app -o=yaml
kubectl get pods -o=wide
```

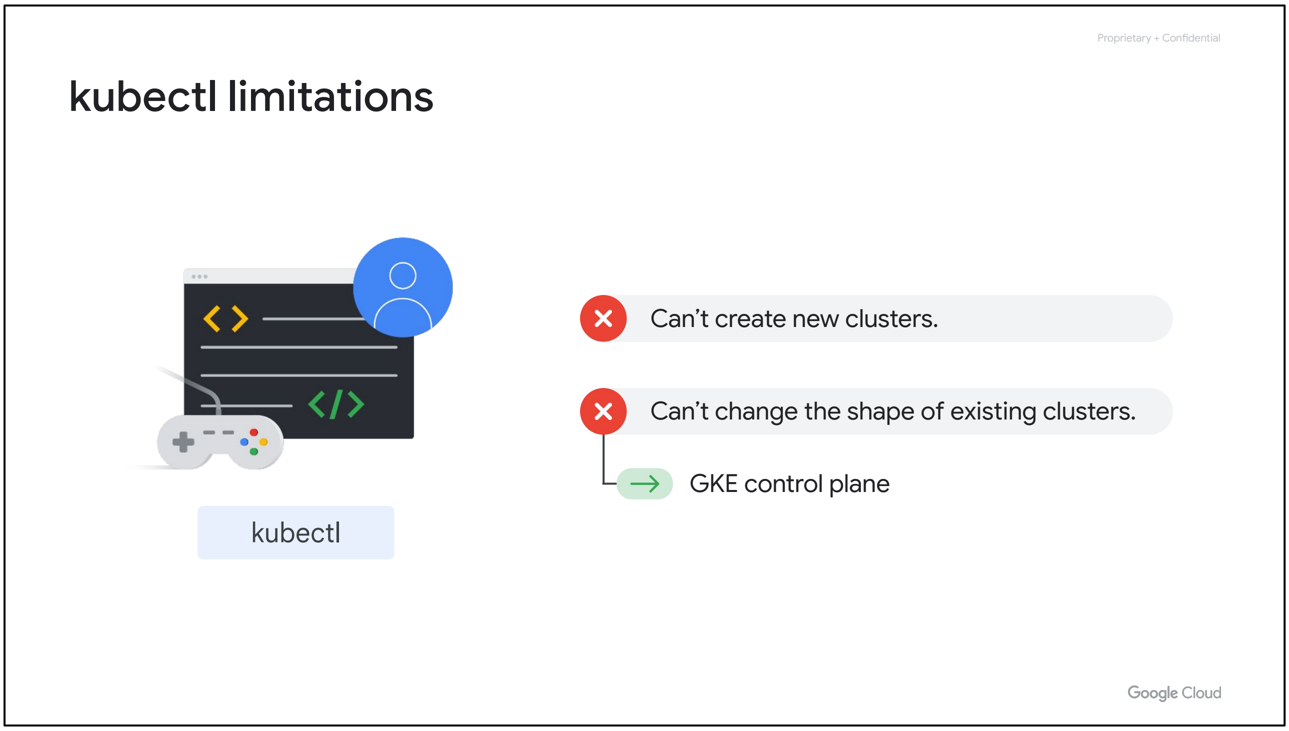
Google Cloud

SAY: The **NAME** specifies the object defined in **TYPE**.

The name field isn't always needed, especially when you use commands that list or show information. For example, if you run the command "kubectl get pods" without specifying a name, the command returns the list of all Pods. To filter this list, specify a Pod's name, such as "kubectl get pod my-test-app" and kubectl will return only information on the Pod named 'my-test-app'.

Some commands let you append **flags** to the end, which you can think of as a way to make a special request. For example, to view the state of a Pod, use the command "kubectl get pod my-test-app -o=yaml". Also, it's worth mentioning that telling kubectl to produce a YAML output can be helpful for other tasks related to Kubernetes objects, for example, recreating an object in another cluster.

Flags can also be used to display additional information. For example, run the command "kubectl get pods -o=wide" to display the list of Pods in "wide" format, which reveals Pods in the list. Wide format also displays which node each Pod is running on.

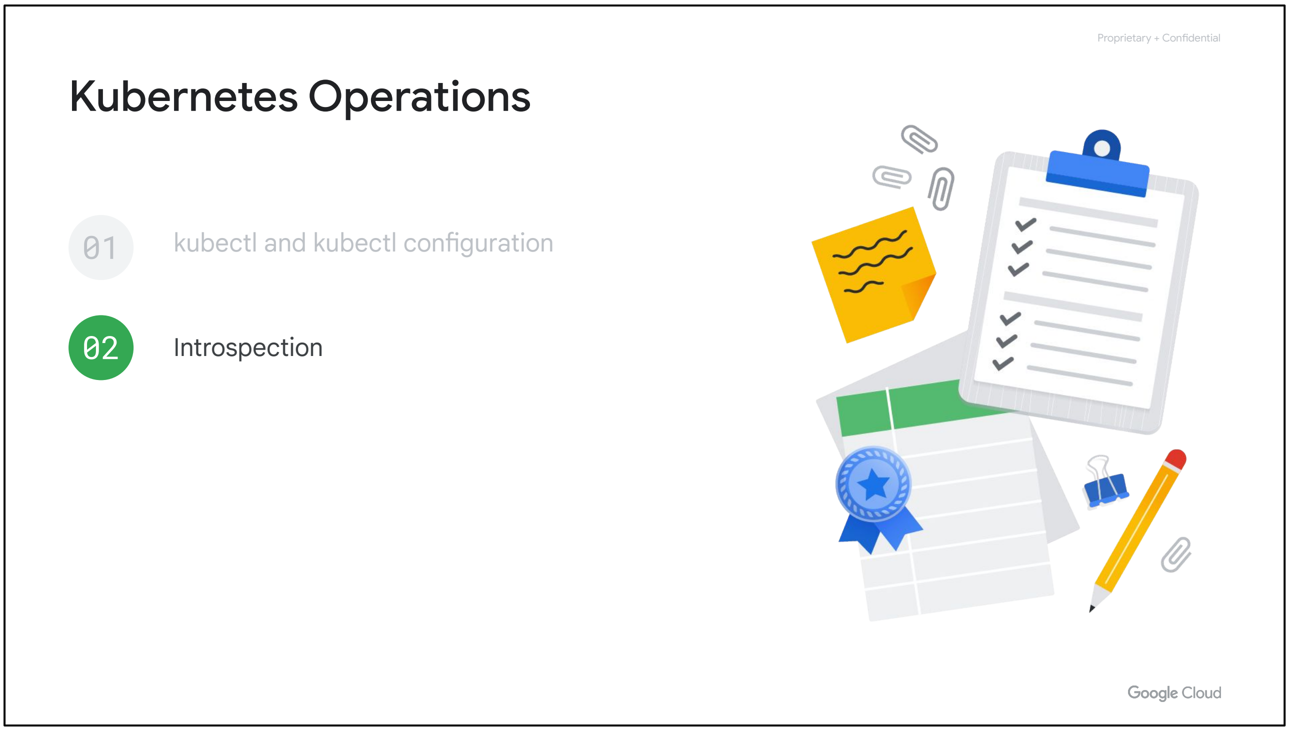


kubectl limitations

- ✗ Can’t create new clusters.
- ✗ Can’t change the shape of existing clusters.
  - → GKE control plane

SAY: Although kubectl is a tool for administering the internal state of an existing cluster, it can’t create new clusters or change the shape of existing clusters. That’s done through the GKE control plane, which the gcloud command and the Google Cloud console interfaces to.

After the config file in the .kube folder is configured, the kubectl command automatically references this file and connects to the default cluster without prompting for credentials.

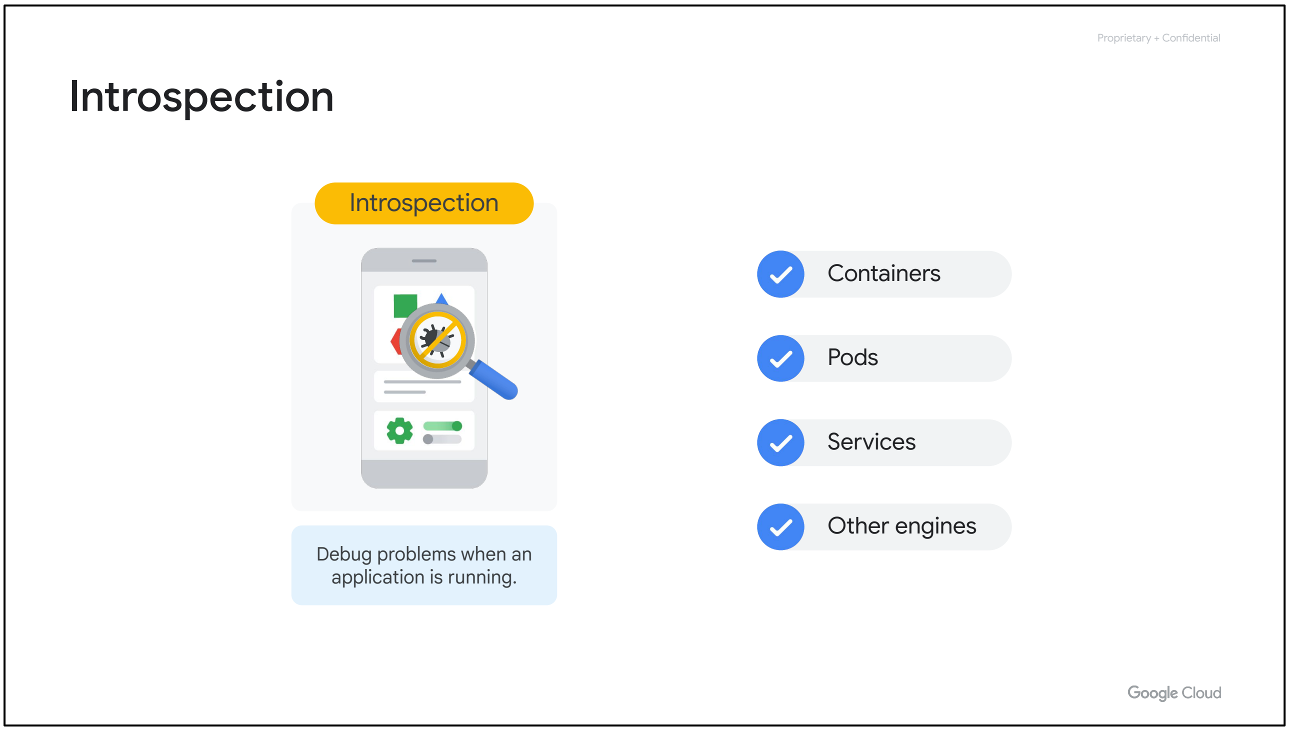


Kubernetes Operations

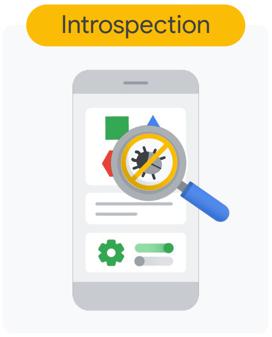
01 kubectl and kubectl configuration

02 Introspection

SAY: Now that you've been introduced to the kubectl command, which is the way to specify the action that you want to perform on a Kubernetes cluster, let's explore how to debug problems when an application is running.



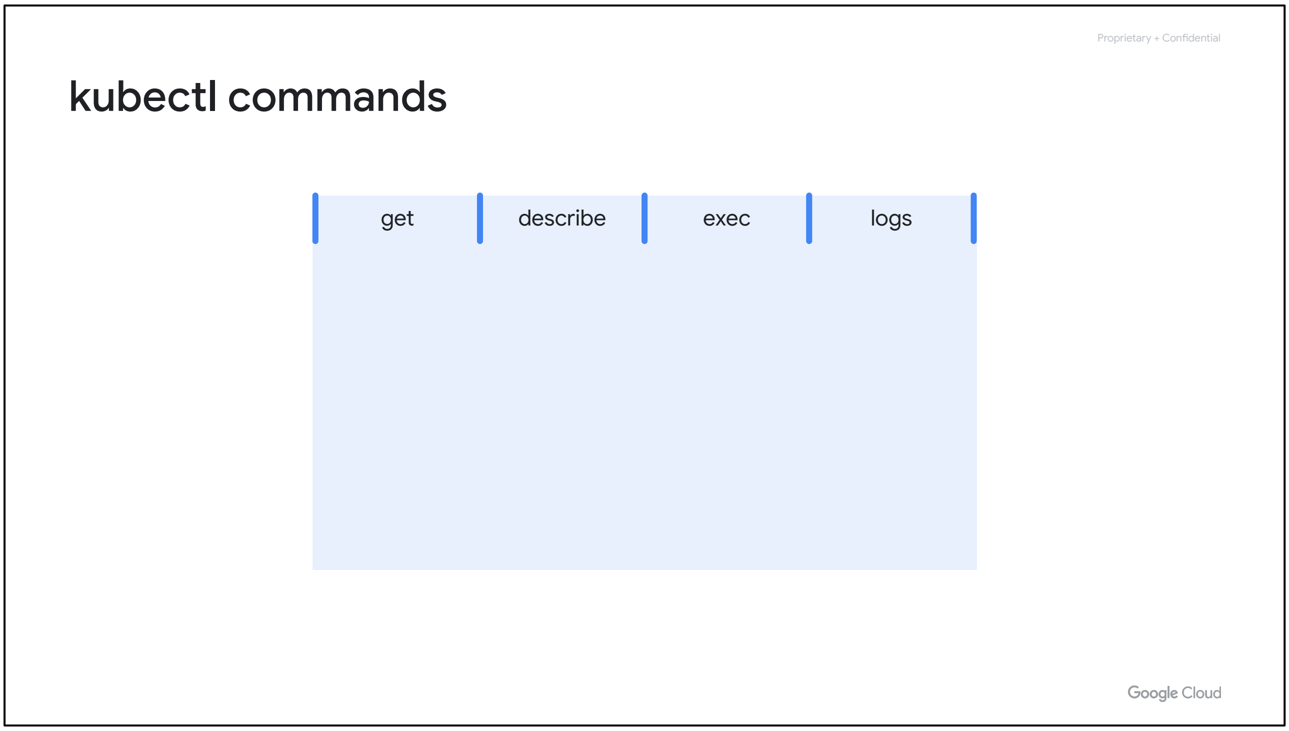
Introspection



Debug problems when an application is running.

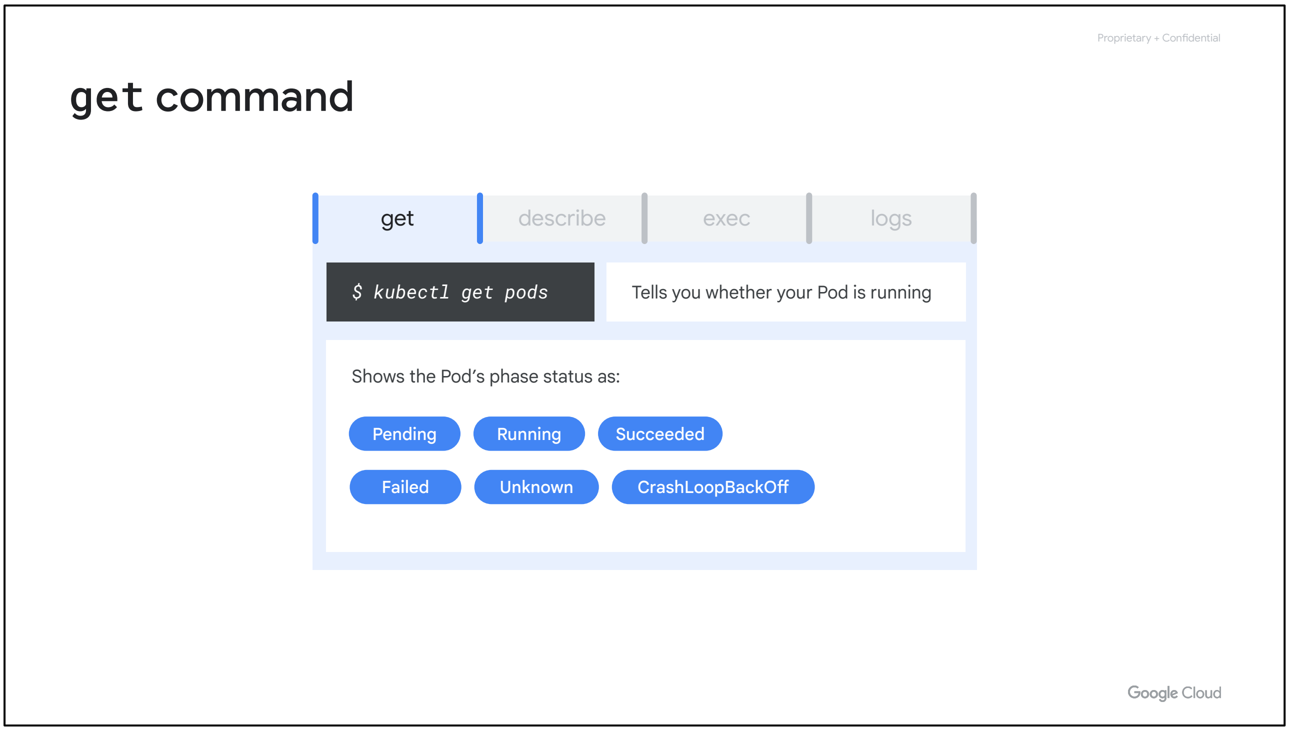
- ✓ Containers
- ✓ Pods
- ✓ Services
- ✓ Other engines

SAY: This process is called **introspection**. It's the act of gathering information about the containers, pods, services, and other engines that run within the cluster.



kubectl commands

SAY: We'll start with four commands to use to gather information about your app: get, describe, exec, and logs.



Proprietary + Confidential

get command

get
describe
exec
logs

```
$ kubectl get pods
```

Tells you whether your Pod is running

Shows the Pod's phase status as:

Pending
Running
Succeeded
Failed
Unknown
CrashLoopBackOff

Google Cloud

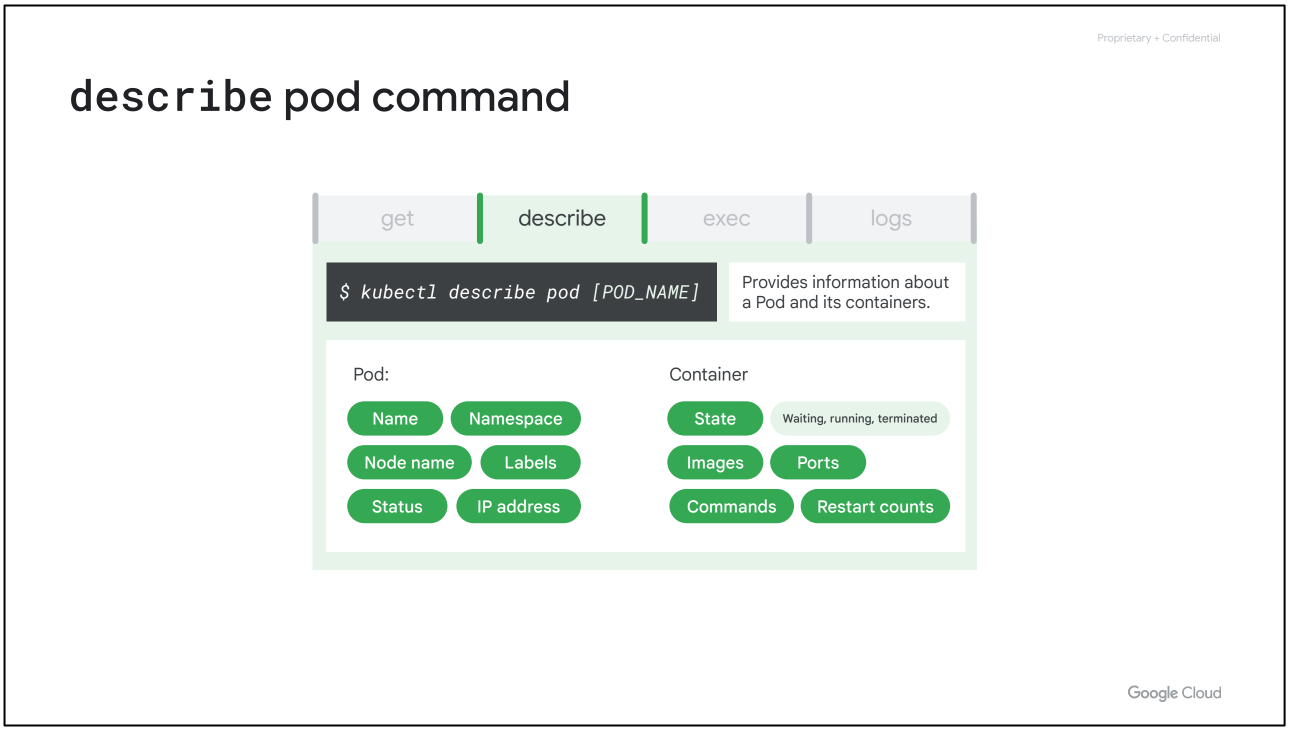
SAY: A good place to start is with Pods, the basic units of Kubernetes. A simple kubectl "`get pods`" command tells you whether your Pod is running. It shows the Pod's phase status as *pending*, *running*, *succeeded*, *failed*, or *unknown*, or *CrashLoopBackOff*.

The phase provides a high-level summary, not the comprehensive details about a Pod or its containers. The **pending** status indicates that Kubernetes has accepted a Pod, but it's still being scheduled. This means that the container images defined for the Pod have not yet been created by the container runtime.

For example, when images are pulled from the repository, the Pod will be in the pending phase. A Pod **runs** after it is successfully attached to a node, and all its containers are created.

Containers inside a Pod can be starting, restarting, or running continuously.

- **Succeeded** means that all containers finished running successfully, or instead, that they terminated successfully and they won't be restarting.
- **Failed** means a container terminated with a failure, and it won't be restarting.
- **Unknown** is where the state of the Pod simply cannot be retrieved, probably because of a communication error between the control plane and a kubelet.
- And **CrashLoopBackOff** means that one of the containers in the Pod exited unexpectedly even after it was restarted at least once. This is a common error. Usually, this means that the Pod isn't configured correctly.



`describe` pod command

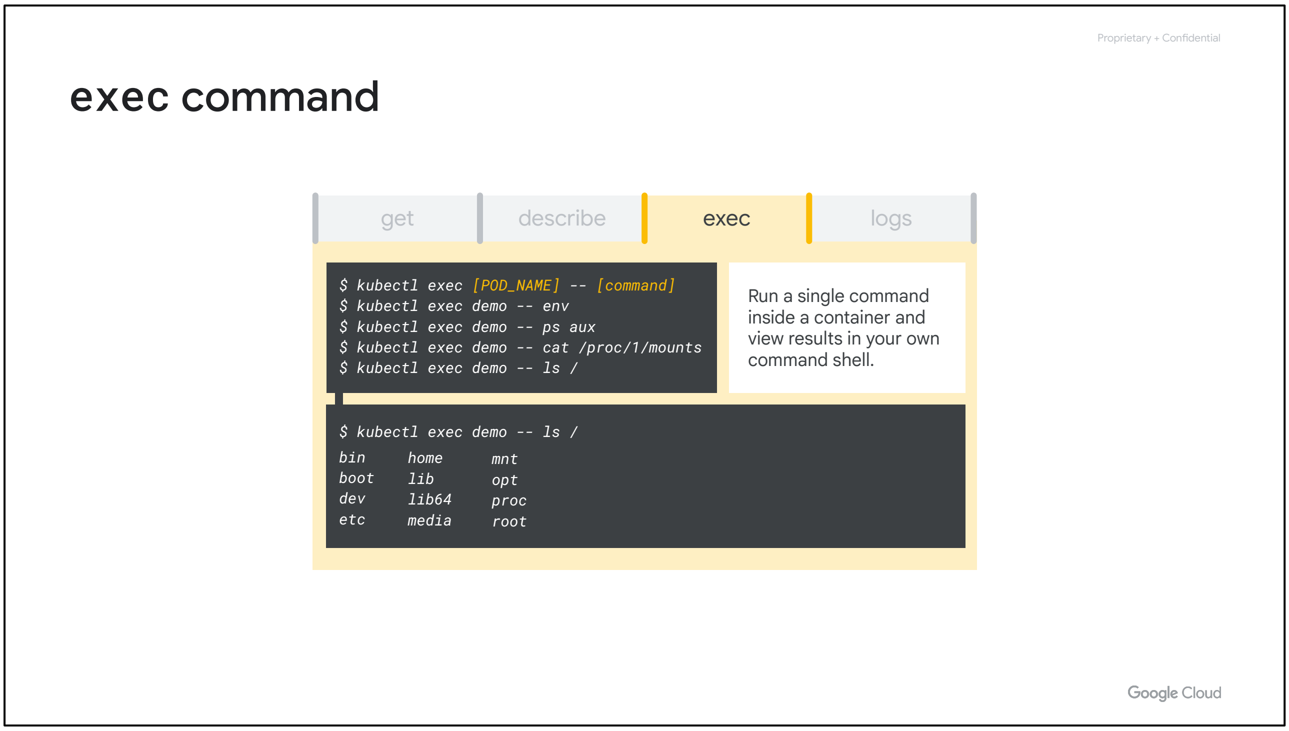
```
$ kubectl describe pod [POD_NAME]
```

Provides information about a Pod and its containers.

SAY: To investigate a Pod in detail, use the kubectl “**`describe`** pod” command.

This command provides information about a Pod and its containers such as labels, resource requirements, and volumes. It also details the status information about the Pod and container. For Pods, the name, namespace, node name, labels, status, and IP address are displayed.

For containers, the state–waiting, running, or terminated, images, ports, commands, and restart counts–are displayed.



exec command

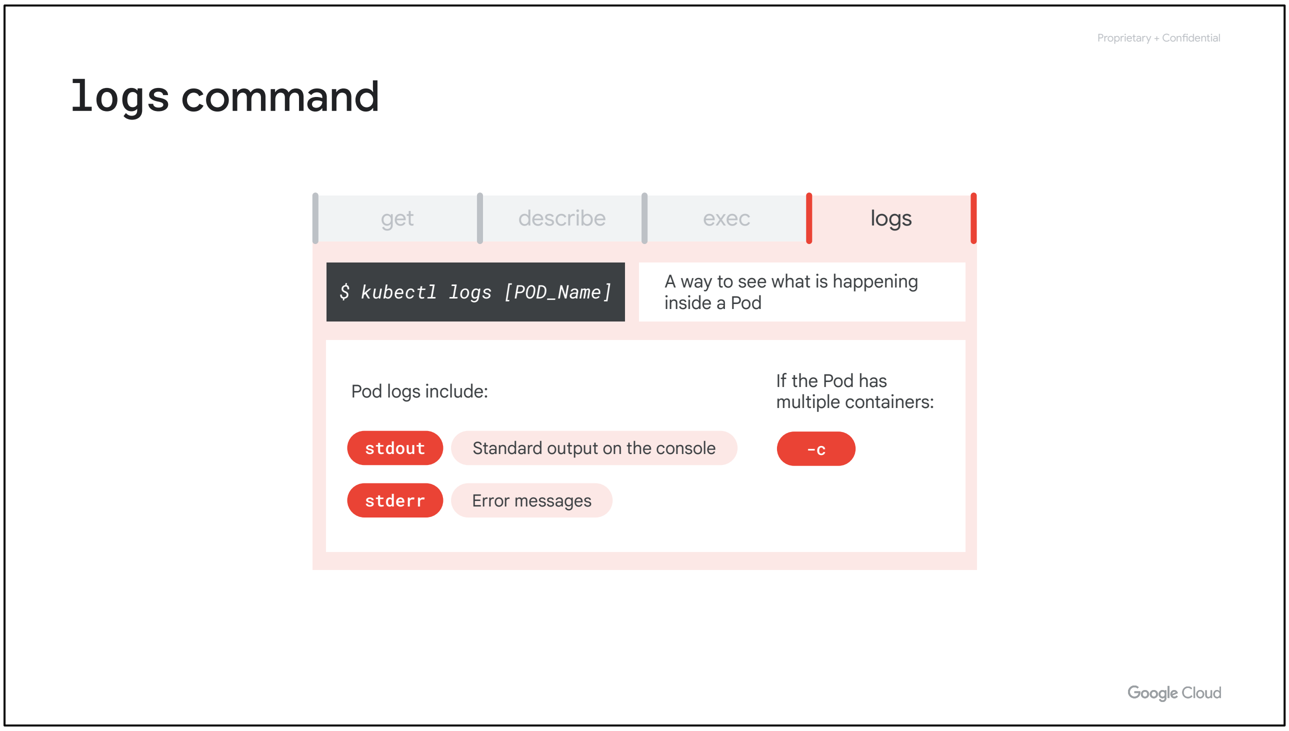
```
$ kubectl exec [POD_NAME] -- [command]
$ kubectl exec demo -- env
$ kubectl exec demo -- ps aux
$ kubectl exec demo -- cat /proc/1/mounts
$ kubectl exec demo -- ls /
```

Run a single command inside a container and view results in your own command shell.

```
$ kubectl exec demo -- ls /
bin     home     mnt
boot    lib      opt
dev     lib64    proc
etc     media    root
```

Google Cloud

SAY: Single command execution using the “**exec**” command lets you run a single command inside a container and view the results in your own command shell.


Proprietary + Confidential

logs command

get | describe | exec | logs

```
$ kubectl logs [POD_Name]
```

A way to see what is happening inside a Pod

Pod logs include:

`stdout`	Standard output on the console
`stderr`	Error messages

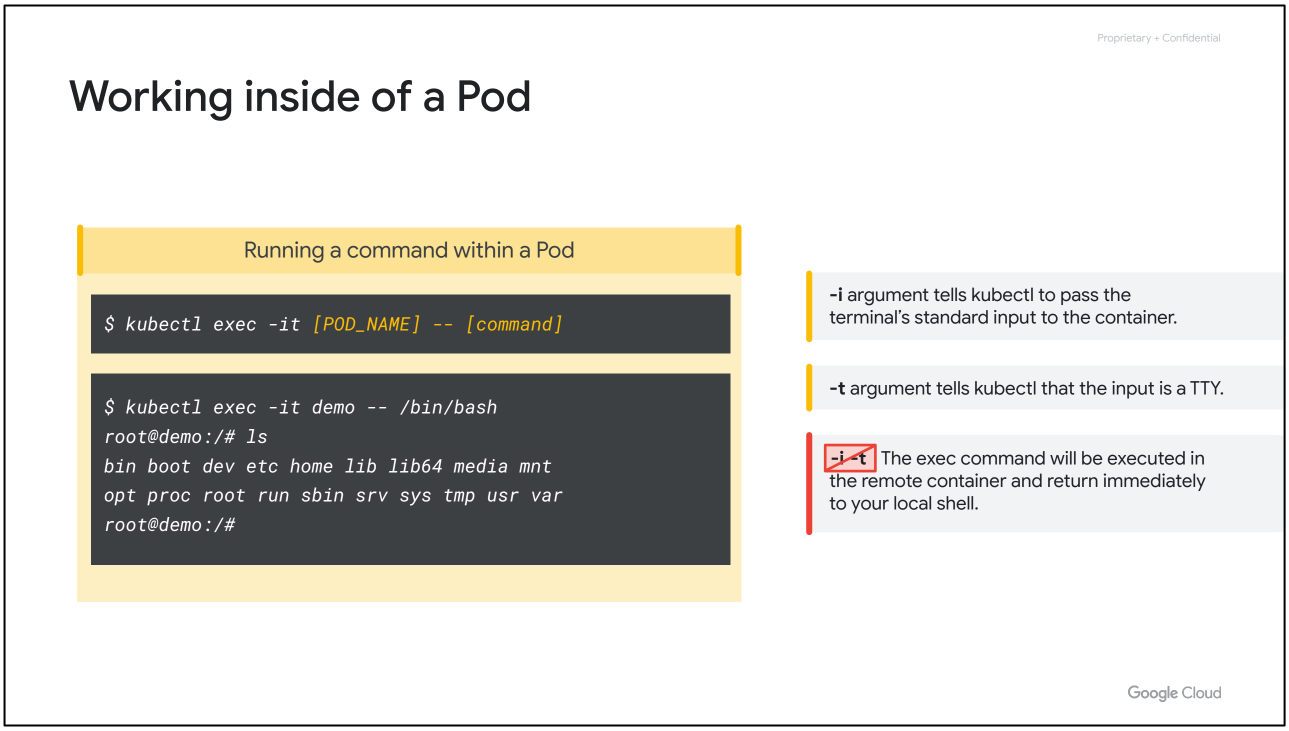
If the Pod has multiple containers: `-c`

Google Cloud

SAY: And finally, the "**`logs`**" command provides a way to see what is happening inside a Pod. This is useful in troubleshooting, as the logs command can reveal errors or debugging messages written by the applications that run inside Pods. The logs contain both the standard output and standard error messages that the applications within the container have generated.

The logs command is useful when you need to find out more information about containers that are failing to run successfully.

And if the Pod has multiple containers, you can use the -c argument to show the logs for a specific container inside the Pod.


Proprietary + Confidential

Working inside of a Pod

Running a command within a Pod

```
$ kubectl exec -it [POD_NAME] -- [command]
```

```
$ kubectl exec -it demo -- /bin/bash
root@demo:/# ls
bin boot dev etc home lib lib64 media mnt
opt proc root run sbin srv sys tmp usr var
root@demo:/#
```

-i argument tells kubectl to pass the terminal’s standard input to the container.

-t argument tells kubectl that the input is a TTY.

~~**-i -t**~~ The exec command will be executed in the remote container and return immediately to your local shell.

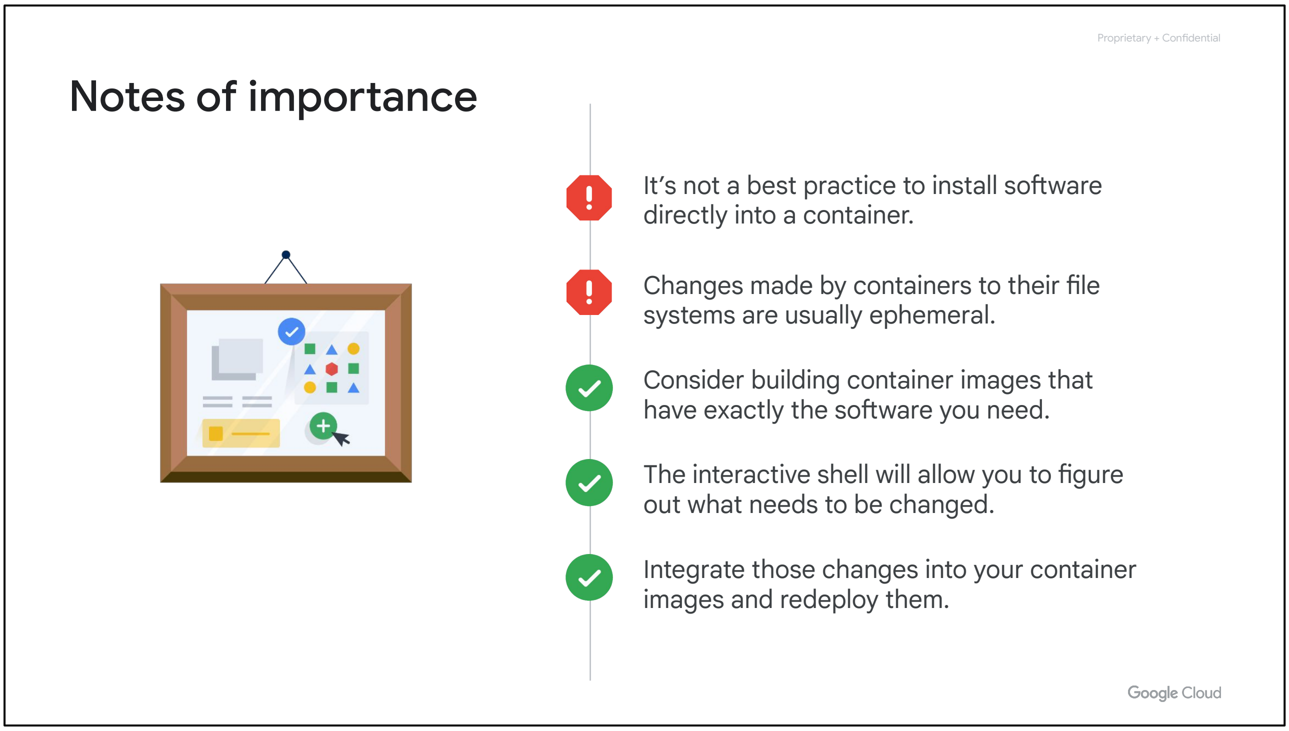
Google Cloud

SAY: There might be a scenario where you need to work inside a Pod, maybe to run a command.

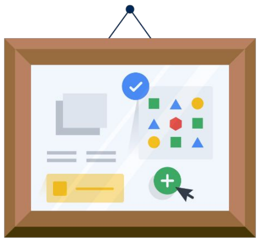
Let’s say you need to install a package, like a network monitoring tool or a text editor, before you can begin troubleshooting. To do so, you can launch an interactive shell using the **`-it switch`**, which connects your shell to the container that allows you to work inside the container.

This syntax attaches the standard input and standard output of the container to your terminal window or command shell.
The **-i** argument tells kubectl to pass the terminal’s standard input to the container, and the **-t** argument tells kubectl that the input is a TTY.

If you don’t use these arguments, then the `exec` command will be executed in the remote container and return immediately to your local shell.



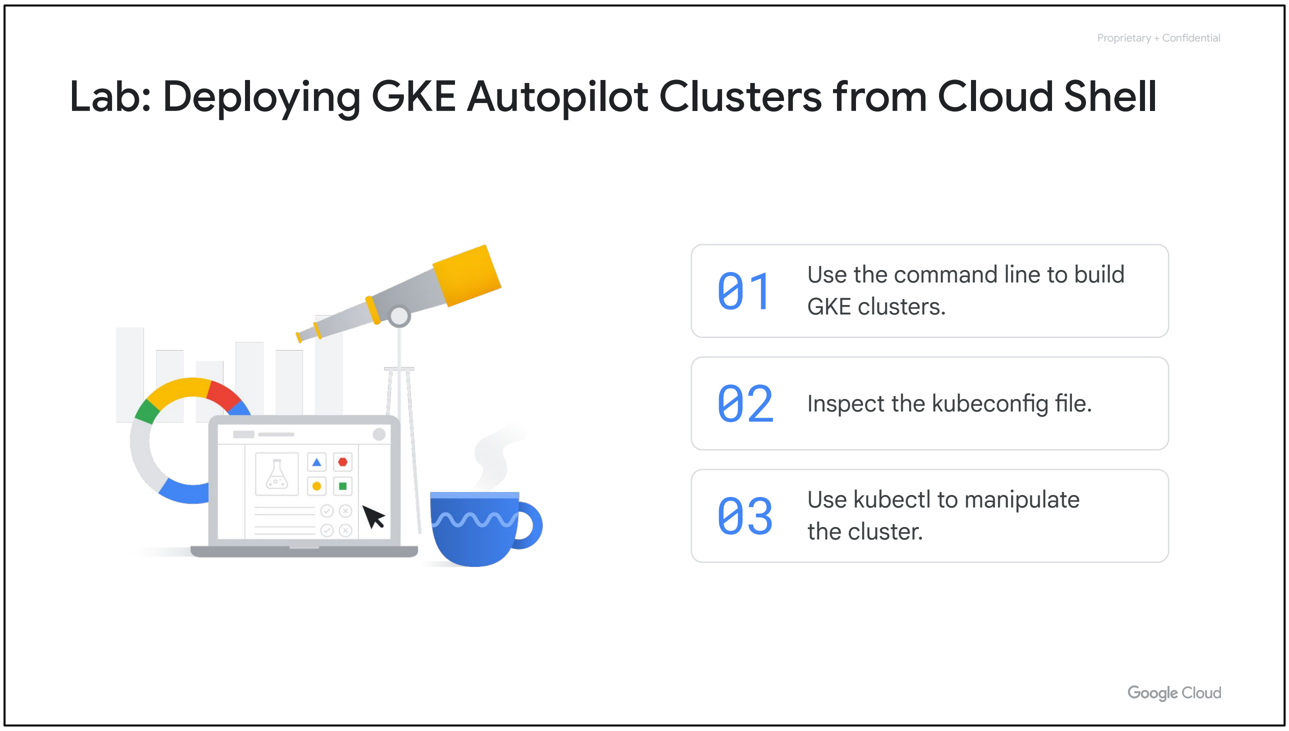
Notes of importance



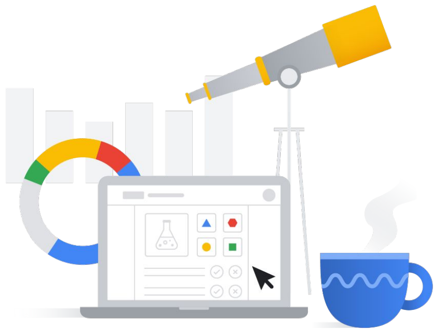
- ❗ It's not a best practice to install software directly into a container.
- ❗ Changes made by containers to their file systems are usually ephemeral.
- ✓ Consider building container images that have exactly the software you need.
- ✓ The interactive shell will allow you to figure out what needs to be changed.
- ✓ Integrate those changes into your container images and redeploy them.

SAY: Now it's important to note that it's not a best practice to install software directly into a container, because changes made by containers to their file systems are usually ephemeral.

Instead, consider building container images that have exactly the software you need, instead of temporarily repairing them at run time. The interactive shell shows you what needs to be changed to solve a problem, but you should then integrate those changes into your container images and redeploy them.



Lab: Deploying GKE Autopilot Clusters from Cloud Shell



01 Use the command line to build GKE clusters.

02 Inspect the kubeconfig file.

03 Use kubectl to manipulate the cluster.

SAY: It's time for the last hands-on lab of this course.

In the lab titled "Deploying GKE Autopilot Clusters from Cloud Shell", you use the command line to build GKE clusters. You'll inspect the kubeconfig file and use kubectl to manipulate the cluster.

Quiz | Question 1 of 4

Question

You want to use kubectl to configure your cluster, but first you must configure it. Where does the kubectl command store its configuration file?

A. The configuration information is stored in the $HOME/.kube/config file.

B. kubectl uses the same authorization and credential tokens as the gcloud CLI utilities.

C. The configuration information is entered in Kubectl before executing commands.

D. The configuration information is stored in environment variables in the current shell when required.

Quiz | Question 1 of 4

Answer

You want to use kubectl to configure your cluster, but first you must configure it. Where does the kubectl command store its configuration file?

A. The configuration information is stored in the $HOME/.kube/config file. ✓

B. kubectl uses the same authorization and credential tokens as the gcloud CLI utilities.

C. The configuration information is entered in Kubectl before executing commands.

D. The configuration information is stored in environment variables in the current shell when required.

Quiz | Question 2 of 4

Question

You attempt to update a container image to a new version by using the `kubectl describe pod` command, but are not successful. The output of the command shows that the Pod status has changed to `Pending`, the state is shown as `Waiting`, and the reason shown is `ImagePullBackOff`.

What is the most probable cause of this error?

A. The container image pull policy has been set to `Never`.

B. The container image failed to download.

C. You specified an invalid container name.

D. The latest container image has already been deployed.

Quiz | Question 2 of 4

Answer

You attempt to update a container image to a new version by using the `kubectl describe pod` command, but are not successful. The output of the command shows that the Pod status has changed to `Pending`, the state is shown as `Waiting`, and the reason shown is `ImagePullBackOff`.

What is the most probable cause of this error?

A. The container image pull policy has been set to `Never`.

[✓ B. The container image failed to download.]

C. You specified an invalid container name.

D. The latest container image has already been deployed.

Quiz | Question 3 of 4

Question

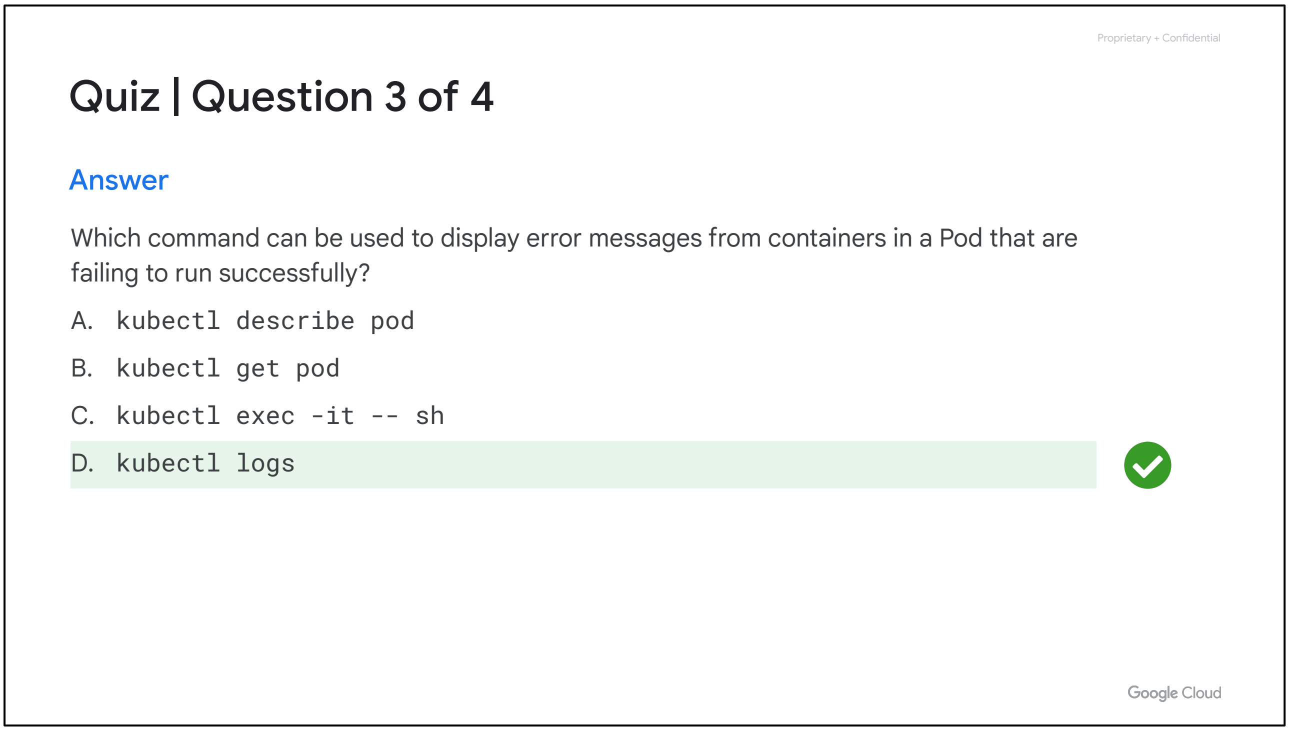
Which command can be used to display error messages from containers in a Pod that are failing to run successfully?

A. `kubectl describe pod`

B. `kubectl get pod`

C. `kubectl exec -it -- sh`

D. `kubectl logs`



Quiz | Question 3 of 4

Answer

Which command can be used to display error messages from containers in a Pod that are failing to run successfully?

A. `kubectl describe pod`

B. `kubectl get pod`

C. `kubectl exec -it -- sh`

[✓ D. `kubectl logs`]

Quiz | Question 4 of 4

Question

What command can be used to identify which containers in a Pod are successfully running, and which are failing or having issues?

A. `kubectl describe pod`

B. `kubectl get pod`

C. `kubectl logs`

D. `kubectl exec`

Quiz | Question 4 of 4

Answer

What command can be used to identify which containers in a Pod are successfully running, and which are failing or having issues?

A. `kubectl describe pod` ✓

B. `kubectl get pod`

C. `kubectl logs`

D. `kubectl exec`